Data formats

for distriburted exchange

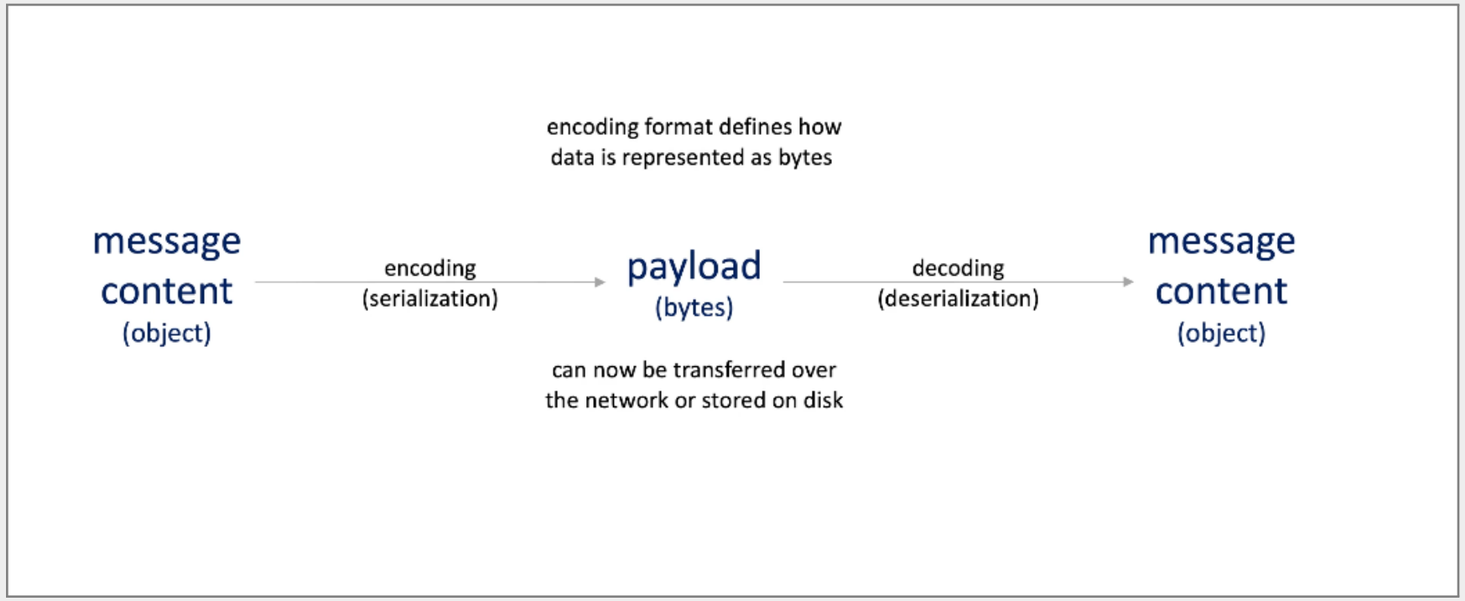
micro-23

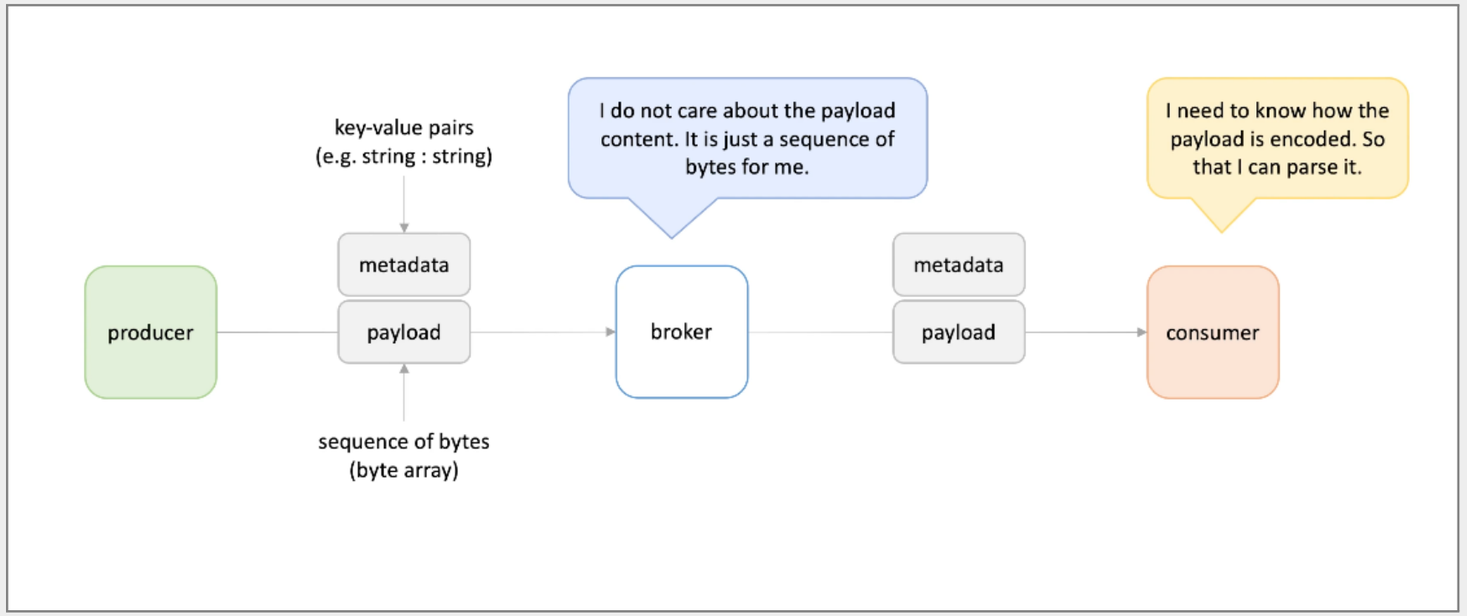
Aliaksei Bialiauski

Designed in LaTeX

> Extensible Markup Language (XML)
>
> JavaScript Object Notation(JSON)
>
> YAML, CSV
>
> Binary data formats: Avro, Protobuf, Thrift

Serialization & Deserialization




key-value pairs
(e.g. string : string)
I do not care about the payload content. It is just a sequence of bytes for me.
I need to know how the payload is encoded. So that I can parse it.
metadata
producer
payload
broker
metadata
payload
consumer
sequence of bytes
(byte array)

Chapter #1:

Extensible Markup Language (XML)

Library in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book id="42">
    <author>Martin Kleppmann</author>
    <title>Designing Data-Intensive Applications</title>
  </book>
  <book id='43'>
    <author>Martin Fowler</author>
    <title>Refactoring</title>
  </book>
</library>
```

XML Based Formats/Protocols

SOAP, RSS, Atom, SVG, XHTML, HTML5,
Open Office XML, XMPP,
SyncML, RDF, XMI, XMIR

XML Schema Definition (XSD)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="book">
    <xs:sequence>
      <xs:element name="author" minOccurs="1" maxOccurs="1"/>
      <xs:element name="title" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:decimal"/>
  </xs:complexType>
  <xs:element name="library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" type="book" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Chapter #2:

JavaScript Object Notation(JSON)

Library in JSON

```
[
  {
    "id": 42,
    "author": "Martin Kleppmann",
    "title": "Designing Data-Intensive Applications"
  },
  {
    "id": 43,
    "author": "Martin Fowler",
    "title": "Refactoring"
  }
]
```

JSON to JavaScript Object and Backwards

```
var a = JSON.parse('{"age": 25}').age;
```

```
JSON.stringify({age: 25});
```

Schema

```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "string",
            "optional": false,
            "name": "io.debezium.data.Uuid",
            "version": 1,
            "default": "00000000-0000-0000-0000-000000000000",
            "field": "id"
          },
          {
            "type": "string",
            "optional": false,
            "field": "type"
          },
          ...
```

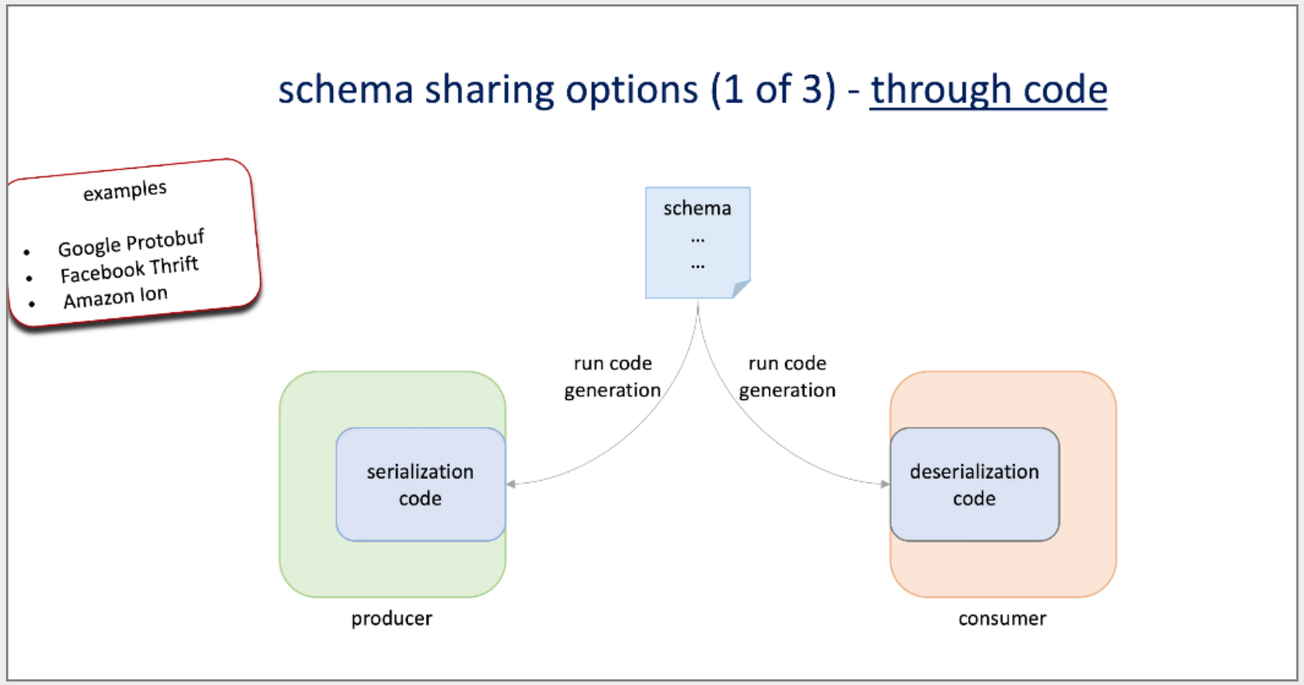
Schema Payload


```
{
  "before": null,
  "after": {
    "id": "8e3a8d8e-8443-4efc-8f23-e3eced22740b",
    "type": "broker",
    "legalCompanyName": "LukasFilm",
    "doingBusinessAs": null,
    "docketMC": null,
    "status": "active",
    "preferredId": null,
    "blockReasonId": null,
    "businessAddressId": "85a52714-b897-494b-bd6c-50d84728e32a",
    "billingInfoId": "e40db062-6b99-4fd6-b916-f8d26e7bbf95",
    "companyId": "06ecb060-1385-419d-91c0-165420773a27",
    "countryOfOrigin": null,
    "taxID": null,
    "created": "2022-07-14T10:27:00.158414Z",
    "updated": "2022-07-14T10:27:00.191556Z"
  }
}
```

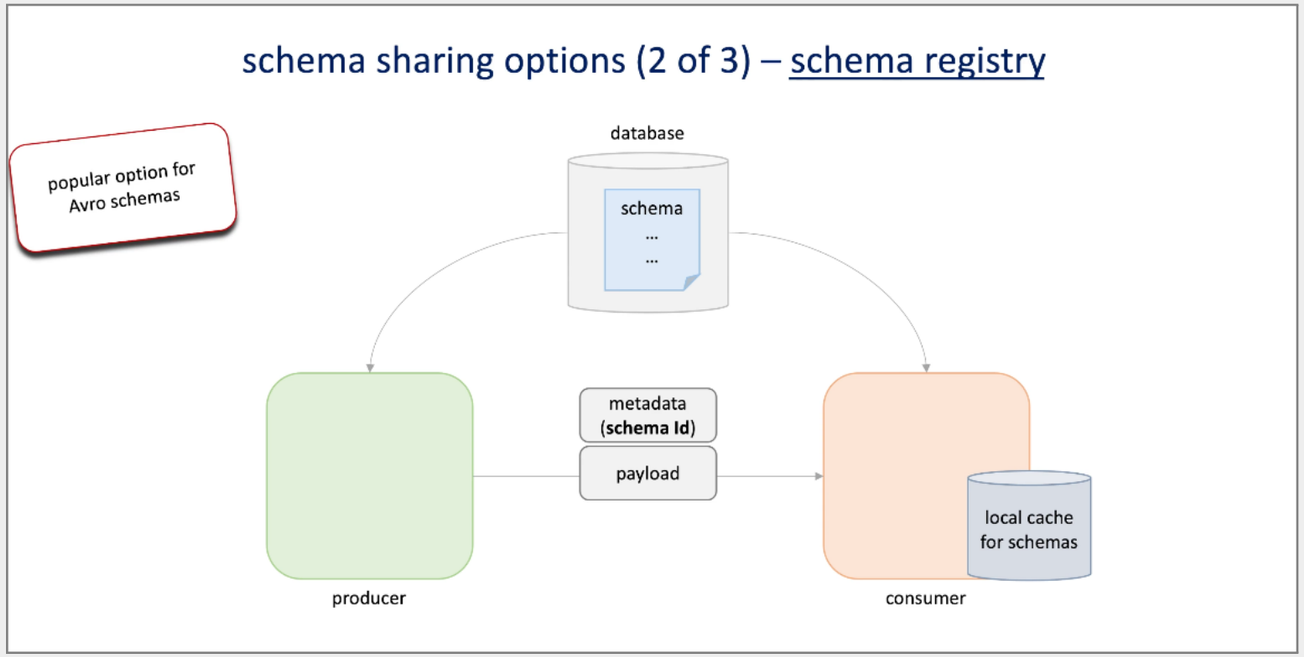
Schema sharing options

- Through code
- Schema registry
- Send along with the payload

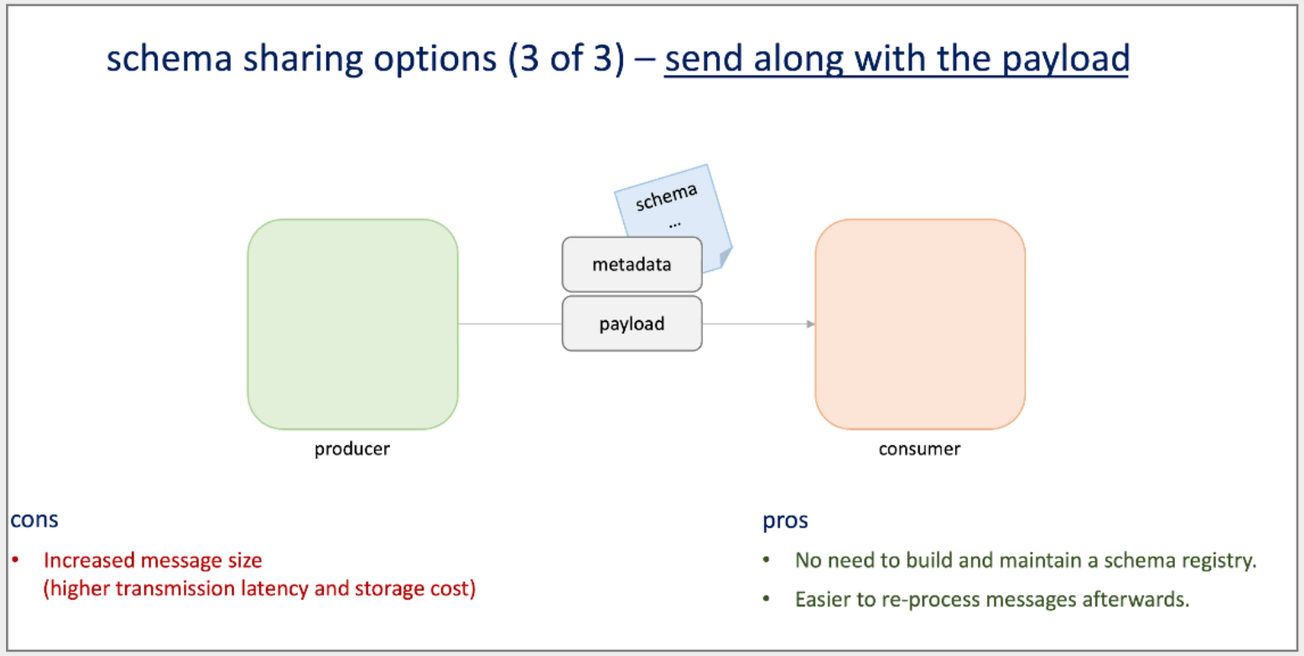
Through code



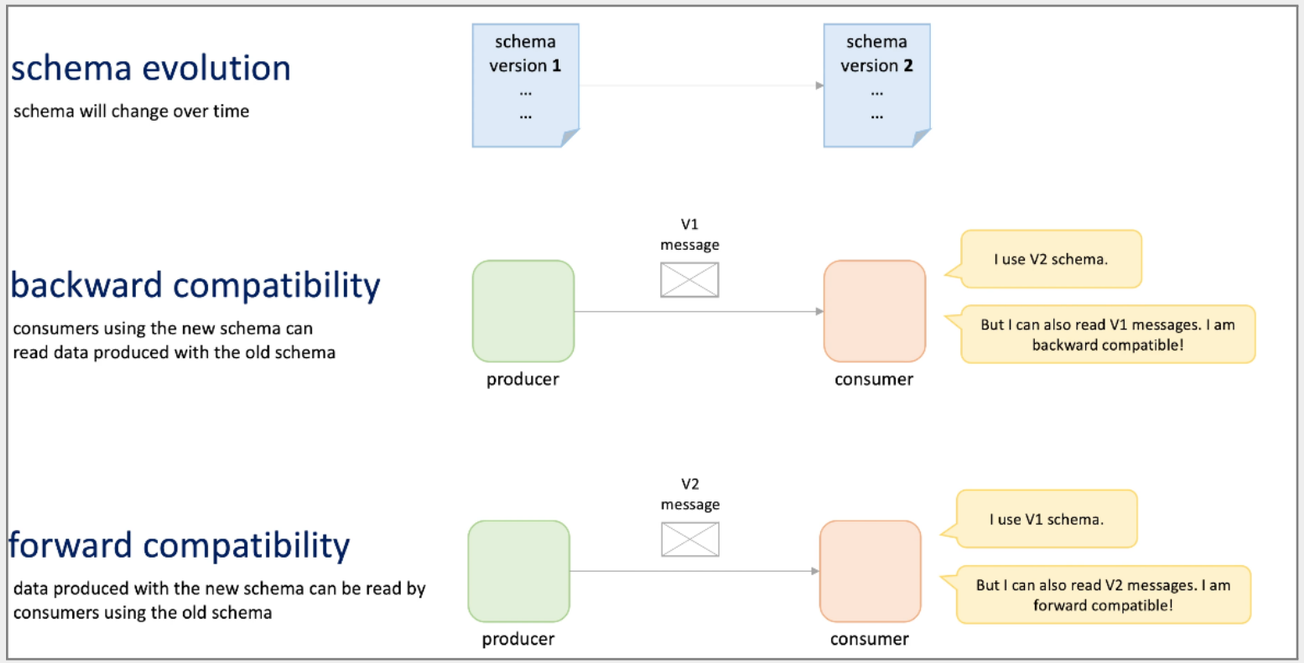
Schema Registry



Send along with payload



Schema evolution



Chapter #3:

YAML, CSV

Yeat Another Markup Language (YAML)

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
    - port: 5432
      targetPort: 5432
  type: LoadBalancer
```

Comma-separated values (CSV)

```
Id,Author,Title
42,Martin Kleppmann,Designing Data-Intensive Applications
43,"Martin Fowler","Refactoring"
```

Chapter #4:

Binary data formats: Avro, Protobuf, Thrift

Avro

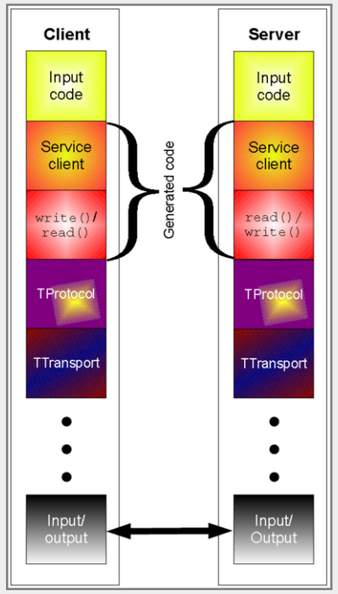
Apache Avro is a data serialization system. It uses JSON for defining data types and protocols, and serializes data in a compact binary format

Protobuf

```
syntax = "proto2";

package tutorial;

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

```
// name
inline bool has_name() const;
inline void clear_name();
inline const ::std::string& name() const;
inline void set_name(const ::std::string& value);
inline void set_name(const char* value);
inline ::std::string* mutable_name();

// id
inline bool has_id() const;
inline void clear_id();
inline int32_t id() const;
inline void set_id(int32_t value);

// email
inline bool has_email() const;
inline void clear_email();
inline const ::std::string& email() const;
inline void set_email(const ::std::string& value);
inline void set_email(const char* value);
inline ::std::string* mutable_email();
```

Thrift



Thrift provides clean abstractions for data transport, data serialization, and application level processing.

Textual vs Binary formats

textual formats

(JSON, XML, CSV, ...)

pros

- Human-readable (easier to debug and test).
- Widely supported by languages and tools.

cons

- Bigger messages (slower to transfer).
- Slower serialization and deserialization.

binary formats

(Thrift, Protobuf, Avro, ...)

pros

- Smaller messages
  (faster to transfer and less space needed to store).
- Faster to serialize/deserialize.

cons

- Not human-readable (harder to debug and test).